UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Análisis de algoritmo de boosting basado en subconjuntos de atributos

Alejandro Rodríguez Rodríguez
Tutor: Gonzalo Martínez Muñoz

Junio 2018

Análisis de algoritmo de boosting basado en subconjunto de atributos

AUTOR: Alejandro Rodríguez Rodríguez
TUTOR: Gonzalo Martínez Muñoz

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2018

Resumen

Este Trabajo Fin de Grado estudiará y describirá diferentes intentos de mejora de dos de los algoritmos más conocidos y efectivos de Boosting y Bagging, el AdaBoost y el RandomForest, los cuales son métodos de aprendizaje automático supervisado que basan su éxito en la agrupación y mejora de clasificadores débiles con el objetivo de obtener un clasificador más robusto, con cuyas estimaciones se obtendrán mejores resultados.

Para ello, se experimentará con la librería Weka basando todas nuestras pruebas en los filtros de selección de atributos que el propio Weka otorga a los usuarios. De tal modo que se realizarán múltiples modificaciones pequeñas en el código fuente de dicho programa, con las cuales se desarrollarán diferentes experimentos con varios conjuntos de datos enfocados a la clasificación que han sido proporcionados por el propio Weka, tales como breast-cancer, diabetes o iris, entre otros.

Por otro lado, se analizarán los filtros de selección de atributos con la finalidad de estudiar su comportamiento sobre los diferentes conjuntos de datos. De esta forma, podremos observar y analizar con mayor exactitud el comportamiento de los algoritmos estudiados a la hora de clasificar un subconjunto de atributos.

Finalmente, una vez efectuados una gran cantidad de experimentos, se analizarán los resultados obtenidos en busca de las posibles mejoras obtenidas y del trabajo futuro que se podrá realizar a raíz de las diferentes investigaciones llevadas a cabo.

Abstract

This Bachelor Thesis will study and describe different attempts to improve two of the best known and most effective Boosting and Bagging algorithms, AdaBoost and RandomForest, which are supervised automatic learning methods that base their success on the grouping and improvement of weak classifiers with the aim of obtaining a more robust classifier, with whose estimates better results will be obtained.

To do this, we will experiment with the Weka library, basing all our tests on the attribute selection filters that Weka itself provides to users. Thus, multiple small modifications will be made to the source code of this program, with which different experiments will be performed with various datasets focused on classification that have been provided by Weka himself, such as breast-cancer, diabetes or iris, among others.

On the other hand, the attribute selection filters will be analyzed in order to study their behavior on the different data sets. In this way, we will be able to observe and analyze with greater accuracy the behavior of the algorithms studied when classifying a subset of attributes.

Finally, once a large number of experiments have been carried out, the results obtained will be analysed in search of the possible improvements obtained and the future work that can be carried out as a result of the different investigations carried out.

Palabras clave

Inteligencia artificial, aprendizaje automático, boosting, bagging, árboles aleatorios, random forest, adaboost, selección de atributos, conjunto de datos.

Keywords

Artifical inteligence, machine learning, boosting, bagging, random tres, random forest, adaboost, feature selectio, dataset.

Agradecimientos

Se lo agradezco a mis padres, por haberme dado la posibilidad de realizar esta carrera y por haberme animado a continuar en los peores momentos.

A Gonzalo, por haberme ayudado y guiado a lo largo de este trabajo.

A la buena gente que he conocido en esta carrera, En especial, a Javi y Sergio, por haber estado ahí desde el primer día de este camino, a Silvia y Elena, por ser unos modelos a seguir desde el momento en el que os conocí, a Patricia, por haber aguantado mi peculiar forma de ser y a Ángel y Óscar, por ser unos genios con los que espero seguir manteniendo el contacto.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Desde que en 1950, Turing desarrollase su famoso test, considerado como el punto de partida de la inteligencia artificial moderna, son muchos los caminos que se han seguido con la finalidad de obtener máquinas capaces de realizar tareas por sí mismas que solo los seres humanos han podido efectuar hasta entonces.

Uno de esos caminos es el aprendizaje automático, en el cual utilizando algoritmos mezclados con razonamiento y autocorrección se pueden conseguir soluciones informáticas que ayudan en multitud de facetas al trabajo humano, desde la fabricación industrial, aumentando las posibilidades de la robótica, a la asistencia sanitaria, mediante diagnósticos de mayor calidad y más rápidos, en los negocios, para descubrir información sobre cómo servir mejor a los clientes, en la educación, mejorando la evaluación de los estudiantes, en las finanzas, proporcionando asesoramiento financiero, en análisis de textos y procesamiento de lenguaje natural y múltiples proyectos que van surgiendo a lo largo de los años.

Es por todo ello, por lo que pretendemos aportar nuestros propios estudios, analizando y observando el comportamiento de un algoritmo de aprendizaje automático, el boosting.

1.2 Objetivos

Nuestros objetivos para este proyecto son los siguientes:

- Objetivo 1. Estudiar el comportamiento del RandomForest y AdaBoost cuando se aplican filtros de selección de atributos.

- Objetivo 2. Estudiar el comportamiento del RandomForest y AdaBoost cuando se elimina progresivamente atributos de los conjuntos de datos.

- Objetivo 3. Comparar los resultados con RandomForest y AdaBoost sin modificar

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Capítulo 1. Introducción, El primer capítulo de la memoria consta de una breve introducción compuesta de una explicación somera de los motivos por los cuales he

realizado este proyecto, los objetivos con los cuales se ha llevado a cabo y la estructura de la memoria.

- Capítulo 2. Estado del arte. En este apartado definimos brevemente cada uno de los conceptos teóricos que engloban este proyecto, en los cuales incluimos, la inteligencia artificial, el aprendizaje automático, los árboles de decisión (ID3 y C4.5), el bagging junto a los random forest y, finalmente, el boosting, con una explicación breve del AdaBoost.
- Capítulo 3. Preparación. Este capítulo indica cuales han sido las diferentes modificaciones llevadas a cabo en el código fuente de Weka y los motivos por los que se han desarrollado. Asimismo, se explicarán los filtros de selección de atributos utilizados.
- Capítulo 4. Experimentos y resultados. En este apartado se mostrarán los resultados obtenidos al haber aplicado dichos filtros y las modificaciones del apartado anterior. Se tratará de analizar cada uno de los resultados.
- Capítulo 5. Conclusiones y trabajo futuro. Finalmente, este capítulo trata de sacar una visión global de los resultados obtenidos en los experimentos anteriores y sus posibles aplicaciones futuras.

2 Estado del arte

2.1 Inteligencia artificial

La inteligencia artificial tiene numerosas definiciones según desde el punto de vista desde el que se estudie. Desde un punto de vista técnico, podemos resumir que la inteligencia artificial es la creación, estudio y utilización de algoritmos con el objetivo de desarrollar máquinas capaces de simular el comportamiento de un ser humano.

Es una rama de la informática cuyo crecimiento en las últimas décadas ha permitido que el ser humano disfrute de numerosas máquinas basadas en ella, tales como los asistentes virtuales de voz, como Cortana, o los robots de limpieza automáticos, entre miles de aparatos de los que disponemos hoy en día.

Las tendencias nos sugieren que muchos sectores tendrán cambios radicales en sus estructuras de uso y consumo gracias a la inteligencia artificial en sus diversas formas. Como ejemplo, se estima que la mayoría de empresas comenzarán a ver como herramientas y análisis de tendencias les van a ayudar en sus decisiones de marketing y también como gracias a la inteligencia artificial pueden establecer criterios de trabajo optimizados para abarcar todos los aspectos relacionados con su negocio, tanto a nivel creativo como de producción o servicio.

2.2 Aprendizaje automático

El aprendizaje automático es una rama de la inteligencia artificial cuyo propósito es otorgar la capacidad de aprender y mejorar el funcionamiento de las máquinas basándose en datos que compilen la experiencia relativa a un problema concreto. Para ello, dichas máquinas tendrán el poder de acceder a un conjunto de datos, con el cual se trabajará con el objetivo de mejorar las decisiones futuras.

Resumidamente, este proceso de aprendizaje automático se basará un proceso de entrenamiento de un conjunto de datos formados por experiencias y estudios pasados, los cuales se analizarán buscando patrones o semejanzas de tal forma que se cree un modelo capaz de tomar buenas decisiones futuras.

Como se puede observar en la figura 2.1, el aprendizaje automático se divide en dos ramas principales de algoritmos: los algoritmos supervisados y los no supervisados.

Los algoritmos de aprendizaje supervisados son aquellos en los que a partir de unos ejemplos previamente etiquetados crean un modelo capaz de asignar una clasificación datos no vistos previamente por el modelo. Los algoritmos no supervisados no disponen de ejemplos previamente etiquetados, por lo que a partir de los datos intenta realizar agrupaciones basándose en su similitud.

Dichas etiquetas corresponden a problemas de regresión en el caso de que sean continuas, mientras que pertenecerán a un problema de clasificación si son de tipo discreto.

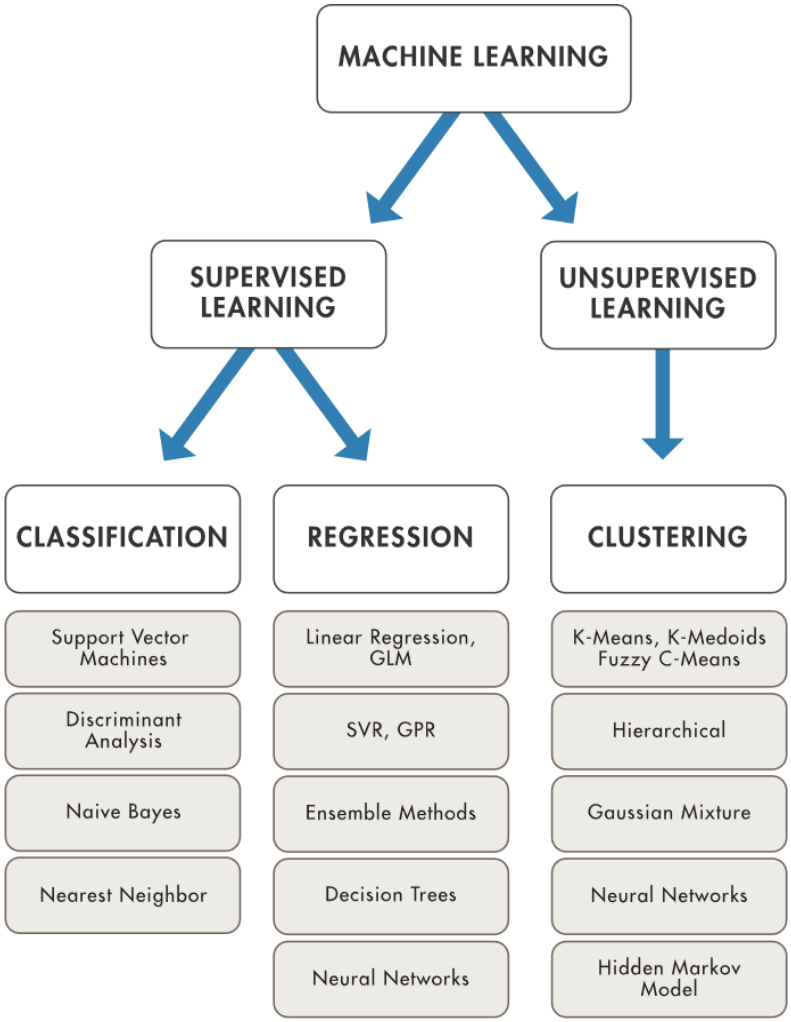

Figura 2-1. Métodos de aprendizaje automático. [https://es.mathworks.com]

En la actualidad, los algoritmos de aprendizaje automático se utilizan para reconocimiento de imágenes y de voz, detección de plagio en textos y vídeos, diagnósticos médicos, robots de chat en centros de llamada, vehículos autónomos, filtros para Spam en el correo electrónico, detección de fraudes en instituciones bancarias y miles de aplicaciones más.

2.3 Arboles de decisión

Un árbol de decisión es un algoritmo de aprendizaje automático supervisado cuyo objetivo se centra principalmente en problemas de clasificación y regresión.

Su funcionamiento se basa en la partición de un conjunto de datos en grupos homogéneos con la finalidad de obtener un mejor pronóstico. Para ello, se crea una estructura de decisiones, similar a un árbol, que comienza con un único nodo que se va ramificando en más nodos a medida que va obteniendo los diferentes resultados

posibles. En la figura 2-2, podemos observar un ejemplo muy sencillo de un árbol de decisión.

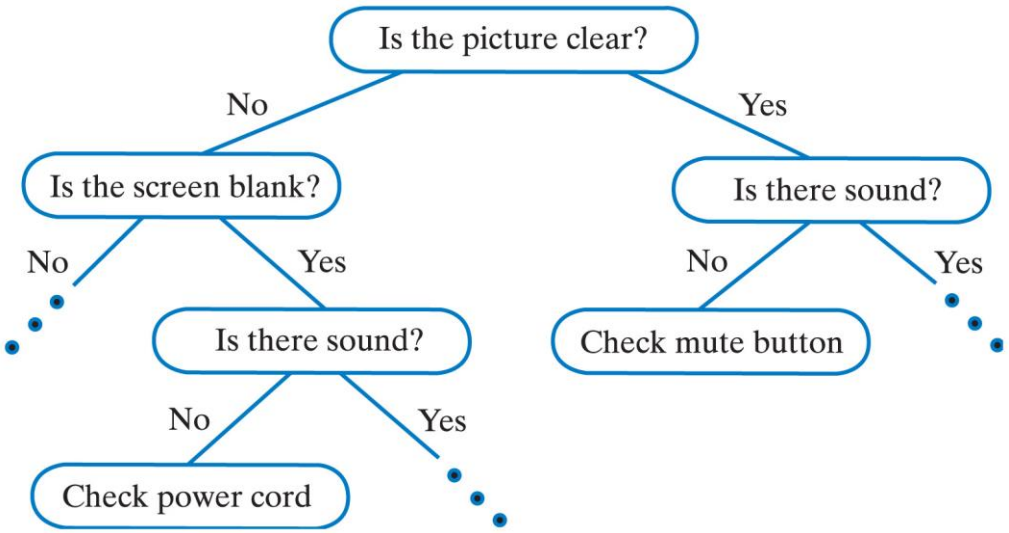

Figura 2-26. Ejemplo de árbol de decisión. [http://www.sfs.uni-tuebingen.de]

2.3.1 ID3

El ID3 (Iterative Dichotomiser 3) es un algoritmo de aprendizaje supervisado desarrollado por Ross Quinlan que consiste en la formación de un árbol de decisión a partir de un conjunto de datos nominales. En estos árboles, la elección de la decisión se establece por medio de la entropía, de tal forma, que se elige, el atributo que mejor ganancia de información proporciona.

Sin embargo, el ID3 contaba con unas limitaciones bastantes importantes, a partir de las cuales Quinlan creó el C4.5 con la intención de arreglarlas. Dichas limitaciones son su imposibilidad de trabajar con valores desconocidos, su pobre manejo con atributos continuos, o su gran sensibilidad con conjuntos de valores extremadamente grandes.

2.3.2 C4.5

El C4.5 es un algoritmo cuyo propósito es generar un árbol de decisión que fue desarrollado por Ross Quinlan con la finalidad de mejorar el algoritmo ID3.

Para cada uno de estos problemas, el C4.5 presenta las siguientes soluciones:

- Capacidad de gestionar valores desconocidos en los datos. Dicha gestión se basa en no utilizar esos valores desconocidos para el cálculo de la entropía de tal forma que se toman como un estado separado.

- Posibilidad de utilizar datos continuos. Se utiliza la normalización a la hora de clasificar, es decir, dado un atributo C con un conjunto de valores continuos, se prueban diferentes particiones de datos en diferentes intervalos y la que mejor resultado obtiene en cuanto a ganancia de información será la utilizada.

- Los árboles de decisión, en ocasiones, tienden a sobreajustar, es decir, la precisión es tan perfecta en la fase de entrenamiento que solo sirve para dicho conjunto de datos. Para ello, se desarrolla la poda, una técnica de eliminación de ramas del árbol de decisión, con la cual no tendremos un precisión tan efectiva, pero resolveremos el problema del sobreajuste, con lo que tendrá mayor efecto en otros conjuntos de datos.

2.4 Bagging

El bagging es un método de aprendizaje automático mediante el cual se generan múltiples versiones de un clasificador débil y agrupan de tal manera para obtener una versión mejorada del conjunto. De esta forma, la versión agrupada predice un promedio cuando se trabaja sobre atributos numéricos y una clase mayoritaria cuando se trabaja con atributos nominales. [3]

Resumidamente, el bagging crea n conjuntos de entrenamiento a partir de un conjunto de datos de entrenamiento inicial por medio de muestreo con reemplazo. Dichos conjuntos serán los que se utilicen para el clasificador final.

Este algoritmo se caracteriza porque obtiene sus mejores resultados cuanto más inestable es el clasificador base, por lo que cuando las diferentes selecciones del conjunto de datos causan resultados dispares, se obtienen unos porcentajes de error realmente efectivos.

2.4.1 Random Forest

Un random forest es una combinación de árboles de decisión en la que los valores de cada árbol dependen de la aleatoriedad, en forma de un vector aleatorio muestreado de la misma forma para todos los árboles del conjunto. [4][9]

Este algoritmo se caracteriza frente a otros métodos de bagging en que el clasificador trabaja con los datos reales del conjunto de datos, pero en forma de particiones. Para ello, cada árbol del Random Forest solo tendrá acceso a un subconjunto aleatorio de atributos a partir de los cuales formará el clasificador final.

2.5 Boosting

El boosting es un método de ensamblado de aprendizaje automático cuyo objetivo es crear un clasificador fuerte a partir de varios clasificadores débiles. Dichos clasificadores débiles son aquellos cuya precisión es ligeramente mejor que un clasificador aleatorio. Un ejemplo de estos es el Decision Stump, un árbol de decisión de un solo nivel, el cual basa su funcionamiento en partir los datos por un único atributo. [5][8]

Este algoritmo basa su funcionamiento en crear un primer modelo a partir del conjunto de datos de entrenamiento, el cual se va mejorando progresivamente construyendo un nuevo modelo que trata de corregir los errores del anterior. Para ello, se van generando

versiones adaptativas del clasificador en las que todos los datos tienen un peso cuyo valor se va actualizando en cada iteración basándose en los peor clasificados, de tal manera que se busca minimizar el error esperado, centrándose en los datos con mayor peso.

2.5.1 AdaBoost

El AdaBoost es un algoritmo de aprendizaje automático creado por Yoav Freund y Robert Schapire que se caracteriza por ser el primer algoritmo fructífero de boosting y que fue desarrollado con la intención de clasificar problemas binarios. [6]

3 Metodología

3.1 Weka

En primer lugar, se explicarán nuestros motivos a la hora de elegir Weka frente a otras librerías de aprendizaje automático, como el scikit-learn. Este último está mejor estructurado y tiene mejor documentación que la mayoría de librerías de aprendizaje automático, pero el factor de que el Weka está escrito en Java y, sobre todo, que cuenta con los filtros de selección de atributos nos han hecho decantarnos por él a la hora de desarrollar este trabajo.

3.1.1 Pre-Análisis

Como ya hemos indicado anteriormente, nuestro objetivo principal en este trabajo es hacer modificaciones en el código de una librería de aprendizaje automático con la finalidad de analizar el comportamiento del boosting ante dichas modificaciones. Estas se centrarán en una adaptación del RandomTree, en el cual seremos capaces de indicarle una secuencia de atributos a evaluar, y en otra alteración de la validación cruzada, en la cual cada conjunto de test podrá convertir todos los valores de x atributos en valor desconocido.

Para ello, se creará un programa que utilice las funciones modificadas y lo compare con el rendimiento del algoritmo de boosting y con el rendimiento del algoritmo de RandomForest sin modificar.

3.1.2 Modificación del código fuente

Random Tree modificado

En primer lugar, se ha procedido a modificar la funcionalidad del RandomTree, en el cual se ha introducido la posibilidad de decidir los atributos que se van a evaluar en cada profundidad del árbol de decisión, con lo que de esta manera ya no es aleatorio, sino que nosotros elegimos el orden y disponibilidad de los atributos, formando un árbol secuencial personalizado.

```
while ((windowSize > 0) && (k-- > 0 || !gainFound)) {

  int chosenIndex = random.nextInt(windowSize);
  attIndex = attIndicesWindow[chosenIndex];

  // shift chosen attIndex out of window
  attIndicesWindow[chosenIndex] = attIndicesWindow[windowSize - 1];
  attIndicesWindow[windowSize - 1] = attIndex;
  windowSize--;

  double currSplit =
    data.classAttribute().isNominal() ? distribution(props, dists,
      attIndex, data) : numericDistribution(props, dists, attIndex,
      totalSubsetWeights, data, tempNumericVals);

  double currVal =
    data.classAttribute().isNominal() ? gain(dists[0], priorVal(dists[0]))
      : tempNumericVals[attIndex];

  if (Utils.gr(currVal, 0)) {
    gainFound = true;
  }

  if ((currVal > val)
    || ((!getBreakTiesRandomly()) && (currVal == val) && (attIndex < bestIndex))) {
    val = currVal;
    bestIndex = attIndex;
    split = currSplit;
    bestProps = props[0];
    bestDists = dists[0];
  }
}
```

Figura 3-7. Código fuente original del funcionamiento del RandomTree

Mediante el siguiente ejemplo, se explicará de una forma más clara:

- Partimos de un conjunto de datos en el cual tenemos 5 atributos (A, B, C, D, E) y una clase (X).

- Por medio de un filtro de selección de atributos, cuyo objetivo es seleccionar cuáles son los mejores atributos del conjunto de datos, determinaremos el orden de atributos que utilizaremos en los árboles de decisión. Un posible orden sería B>C>A>E>D, donde el atributo B sería el que más información aporta y el atributo D el que menos.

- Dicho orden lo pasamos tanto en el Adaboost (configurado para utilizar RandomTrees como clasificador base) como en el RandomForest para que sea utilizado en sus árboles de decisión.

- En el RandomTree, se analizará si hay un orden modificado o simplemente tiene que funcionar de forma aleatoria, como un árbol aleatorio corriente.

- En caso de que se haya un determinado orden, en cada nodo del árbol se analizarán solo los primeros K+1 atributos de dicho orden teniendo K como la profundidad actual del árbol, es decir, si estamos en el primer nodo

(profundidad 0), solo podremos analizar el primer atributo, que coincide con el más relevante según el filtro. En el ejemplo anterior sería el atributo B. En el segundo nodo (profundidad 1), ya podremos analizar dos atributos, el B y el C, en el cual se escogerá el que mejores pesos y ganancias tenga. En el tercer nodo igual, y así sucesivamente hasta que finalmente se termina el árbol.

- De esta forma, se podrá dar el caso de que los peores atributos no se utilicen en ningún momento.

```
int k = 0;
//k--;
while (k <= depth && !gainFound) {
  attIndex = attIndicesWindow[k];

  double currSplit =
    data.classAttribute().isNominal() ? distribution(props, dists,
      attIndex, data) : numericDistribution(props, dists, attIndex,
      totalSubsetWeights, data, tempNumericVals);

  double currVal =
    data.classAttribute().isNominal() ? gain(dists[0], priorVal(dists[0]))
      : tempNumericVals[attIndex];

  if (Utils.gr(currVal, 0)) {
    gainFound = true;
  }

  if ((currVal > val)
    || ((!getBreakTiesRandomly()) && (currVal == val) && (attIndex < bestIndex))) {
    val = currVal;
    bestIndex = attIndex;
    split = currSplit;
    bestProps = props[0];
    bestDists = dists[0];
  }
  k++;
}
```

Figura 3-2. Código fuente modificado del funcionamiento del RandomTree

Esto a primera vista, nos lleva a pensar que mejorará la precisión de las ejecuciones de los algoritmos, ya que podemos elegir cuál es el mejor atributo en cada selección y eliminaremos el posible ruido que nos proporcionen los diferentes atributos, ya que se puede dar el caso de que un atributo no solo no aporte nada a las predicciones, sino que además introduzca información errónea.

Poda basada en los filtros de selección de atributos

Por otro lado, teniendo en cuenta las modificaciones anteriores, procederemos a modificar el funcionamiento de la validación cruzada con la finalidad de observar cuál es el comportamiento de los algoritmos por cada filtro de selección de atributos en el caso de que a cada conjunto de test le removamos un atributo por cada partición de la validación cruzada.

Para ello, hemos introducido otro vector similar al anterior de tal forma que, por cada iteración de la validación cruzada, pondremos todos los valores de un atributo a valor desconocido, representado por "?".

En estas pruebas, hemos optado por utilizar unos vectores de orden inversos a los obtenidos para las pruebas anteriores, por lo que presumiblemente se observará un ligero retroceso en los resultados de los algoritmos por cada atributo que se anula.

```
// Do the folds
for (int i = 0; i < numFolds; i++) {
  Instances train = data.trainCV(numFolds, i, random);
  setPriors(train);
  Classifier copiedClassifier = AbstractClassifier.makeCopy(classifier);
  copiedClassifier.buildClassifier(train);
  if (classificationOutput == null && forPrinting.length > 0) {
    ((StringBuffer)forPrinting[0]).append("\n=== Classifier model (training fold " + (i + 1) +") ===\n\n" +
            copiedClassifier);
  }
  Instances test = data.testCV(numFolds, i);
  if(this.m_MissingOrder != null) {
      for(int k = 0; k < this.m_MissingOrder.size(); k++) {
        Attribute att = test.attribute(this.m_MissingOrder.get(k));
        test.deleteAttributeAt(this.m_MissingOrder.get(k));
        test.insertAttributeAt(att, this.m_MissingOrder.get(k));
      }
      //System.out.print(this.m_MissingOrder.toString());
      //System.out.print(test.toString());
  }
  if (classificationOutput != null){
    evaluateModel(copiedClassifier, test, forPrinting);
  } else {
    evaluateModel(copiedClassifier, test);
  }
}
```

Figura 3-3. Código fuente modificado del funcionamiento de la validación cruzada

3.2 Filtros de selección de atributos

A la hora de determinar los diferentes ordenes de evaluación de atributos en los árboles de decisión que han sido utilizados en las modificaciones del código fuente, nos hemos basado en los diferentes filtros de selección de atributos existentes en el propio Weka.

3.2.1 Infogain

InfoGain calcula la ganancia de información de cada atributo para la clase. Los valores van desde 0 hasta 1 como máximo, y cuanto más puntuación tenga, significa que más información provee a la clase resultante.

3.2.2 Correlation

Correlation se basa en el coeficiente de correlación de Pearon, la cual es una medida de relación lineal entre dos variables, en este caso entre un atributo y la clase. Sus valores pueden ir desde 1 hasta -1.

$$\rho_{X,Y} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

3.2.3 OneR

OneR (One Rule) es un método simple, pero efectivo en el cual se genera una regla para cada atributo y se selecciona la que menor error nos muestre, es decir, se genera un árbol de decisión de un solo nodo, con el cual se escoge la One Rule que tenga menor tasa de error.

3.2.4 ReliefF

ReliefF evalúa una instancia repetidamente y la compara con las instancias vecinas, de tal forma que si dichas instancia tiene un valor diferente con la misma clase resultante, esta es penalizada.

3.2.5 Wrapper

WrapperSubset es un método bastante efectivo de selección de atributos en el cual por medio de un algoritmo de aprendizaje se evalúan diferentes selecciones de atributos con el objetivo de ordenador y descartar los atributos innecesarios. Cabe decir que los resultados variarán según se utilice un algoritmo u otro, no tendremos el mismo resultado con el árbol J48 que con un RandomTree normal.
En nuestro caso, hemos decidido utilizar el J48 para las pruebas.

3.2.6 Cfs

Cfs es otro algoritmo de subselección de atributos cuyo funcionamiento es similar al anteriormente mencionado Correlation, con la característica de que este poda los atributos que considera innecesarios.

3.2.7 Classifier

Classifier es un método de subselección de atributos en el cual se utiliza un clasificador para estimar la importancia de cada atributo utilizando diferentes selecciones de atributos del conjunto de datos. Al igual que en el Wrapper, hemos decidido utilizar el J48.

4 Experimentos y resultados

4.1 Programa de pruebas

Para los experimentos que se van a realizar hemos creado un main escrito en Java, en el cual prepararemos los diferentes experimentos llevados a cabo, incluyendo la utilización de varios filtros de selección de atributos, la validación cruzada, las evaluaciones y las diferentes gráficas comparativas entre RandomForest y AdaBoost.

Para todos ellos, utilizaremos una validación cruzada de 10 particiones con la misma semilla de aleatoriedad con la finalidad de trabajar con las mismas particiones para todas las pruebas.

Para nuestras pruebas, hemos decidido utilizar una selección de algunos de los conjuntos de datos más conocidos para problemas de clasificación, entre los cuales incluimos los siguientes:
Breast-cancer, credit-g, diabetes, glass, ionosphere, iris, soybean, vote y waveform.

Todos estos experimentos seguirán el mismo patrón para todos los casos posibles:

1. Aplicación de un filtro de selección de atributo En concreto se han utilizado:Correlation, OneR, ReliefF, WrapperSubset, CfsSubset y ClassifierSubset.
2. Fase entrenamiento con y sin modificaciones del RandomTree
3. Test de los algoritmos a estudiar (AdaBoost y RandomForest modificados) con gráficas comparativas resultantes.

Filtro de selección de atributos

```
AttributeSelection selector = new AttributeSelection();
InfoGainAttributeEval evaluator = new InfoGainAttributeEval();
Ranker ranker = new Ranker();
ranker.setNumToSelect(Math.min(500, trainData.numAttributes() - 1));
selector.setEvaluator(evaluator);
selector.setSearch(ranker);
selector.SelectAttributes(trainData);

newOrder = selector.selectedAttributes();
newOrder = Arrays.copyOf(newOrder, newOrder.length-1);
System.out.println("InfoGain Order: "+ Arrays.toString(newOrder));
```

Figura 4-8. Ejemplo de utilización del filtro de selección atributos.

Evaluación de Random Forest y AdaBoost

```
rf = new RandomForest(newOrder);
a1 = new AdaBoostM1(newOrder, false);
for(int j = lengthOrder; j > -1; j--) {
    Evaluation evaluation;
    start = Instant.now();

    means = new double[iterations];
    mean = 0;

    trainDataAux = new Instances(trainData);
    evaluation = new Evaluation(trainDataAux);
    if(j != lengthOrder) {
        lengthAux.add(newOrder[j]);
        evaluation.setMissingOrder(lengthAux);
    }

    for(int i = 0; i < iterations; i++) {
        evaluation.crossValidateModel(rf, trainDataAux, 10, new Random(1));
        means[i] = evaluation.errorRate()*100;
        mean += evaluation.errorRate()*100;
    }
    mean = mean/means.length;

    varianza = 0;
    for(int i = 0; i < means.length; i++) {
        varianza += Math.pow((means[i]-mean), 2);
    }
    varianza = varianza/means.length;

    stddev = 0;
    stddev = Math.sqrt(varianza);

    end = Instant.now();

    System.out.println("Gain-" + j + " (RandomForest): "+ mean + " ± "+ stddev);
    timeElapsed = Duration.between(start, end);
    System.out.println("Time taken: "+ timeElapsed.toMillis() +" milliseconds");
    if(j == lengthOrder)
        totalMeansRF[1] = mean;
    totalIncrementalRF[k] = mean;
```

Figura 4-9. Ejemplo de evaluación de medidas de error y tiempo.

Obtención de grafica comparativa

```
xylineChart = ChartFactory.createXYLineChart(fileName, "IncrementalFeatureSelection", "Mean", createIncrementalDataset(),
        PlotOrientation.VERTICAL, true, true, false);

chartPanel = new ChartPanel(xylineChart);
chartPanel.setPreferredSize(new java.awt.Dimension(750, 500));

plot = xylineChart.getXYPlot();
range = (NumberAxis) plot.getRangeAxis();
range.setRange((Arrays.stream(totalIncrementalRF).min().getAsDouble() - 5), (Arrays.stream(totalIncrementalAB).max().getAsDouble() + 5));
range.setTickUnit(new NumberTickUnit(0.5));
//domain.setVerticalTickLabels(true);


renderer = new XYLineAndShapeRenderer();
renderer.setSeriesPaint(0, Color.RED);
renderer.setSeriesPaint(1, Color.BLUE);
//renderer.setSeriesPaint(2, Color.ORANGE);
renderer.setSeriesStroke(0, new BasicStroke(1.5f));
renderer.setSeriesStroke(1, new BasicStroke(1.5f));
//renderer.setSeriesStroke(2, new BasicStroke(1.5f));
plot.setRenderer(renderer);

dateFormat = new SimpleDateFormat("yyyyMMddHHmm");

ChartUtilities.saveChartAsPNG(new File("./"+ fileName + "(InfoGain)" + dateFormat.format(new Date()) +".png"), xylineChart, 750, 500);
```

Figura 4-10. Ejemplo de obtención de gráfica comparativa.

4.2 Resultados

4.2.1 Comparativas de selecciones de atributos

En primer lugar, se muestran las diferentes gráficas resultantes a la hora de variar la selección de atributos utilizando los algoritmos originales en el primer caso, seguido de las diferentes selecciones de atributos detalladas anteriormente.
En la figura 4-4, 4-5, 4-6, 4-7 y 4-8, se puede observan las distintas gráficas comparativas obtenidas, en las cuales podemos ver el porcentaje de error obtenido para cada uno de los conjuntos de datos utilizados con diferentes filtros de selección de atributos.

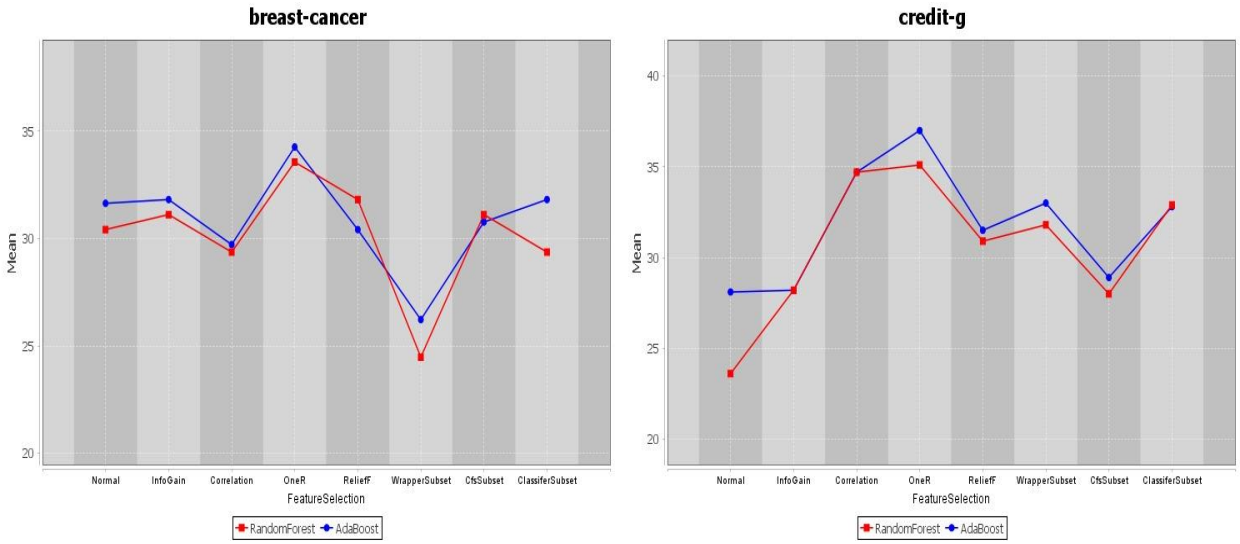

Figura 4-4. Gráfica comparativa de breast-cancer y credit-g.

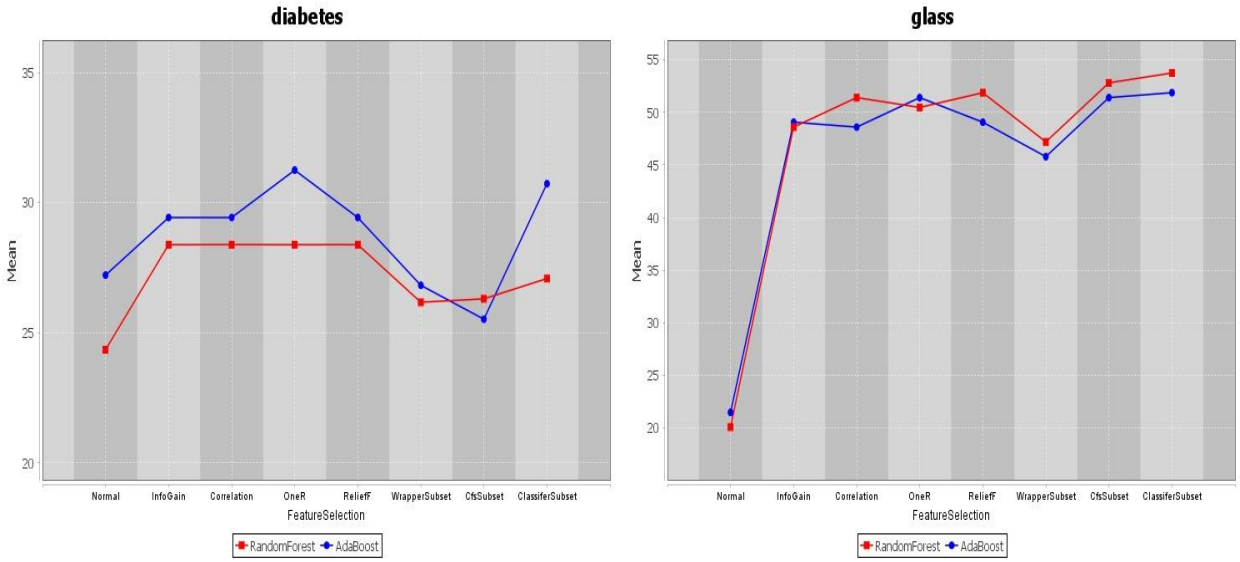

Figura 4-5. Gráfica comparativa de diabetes y glass.

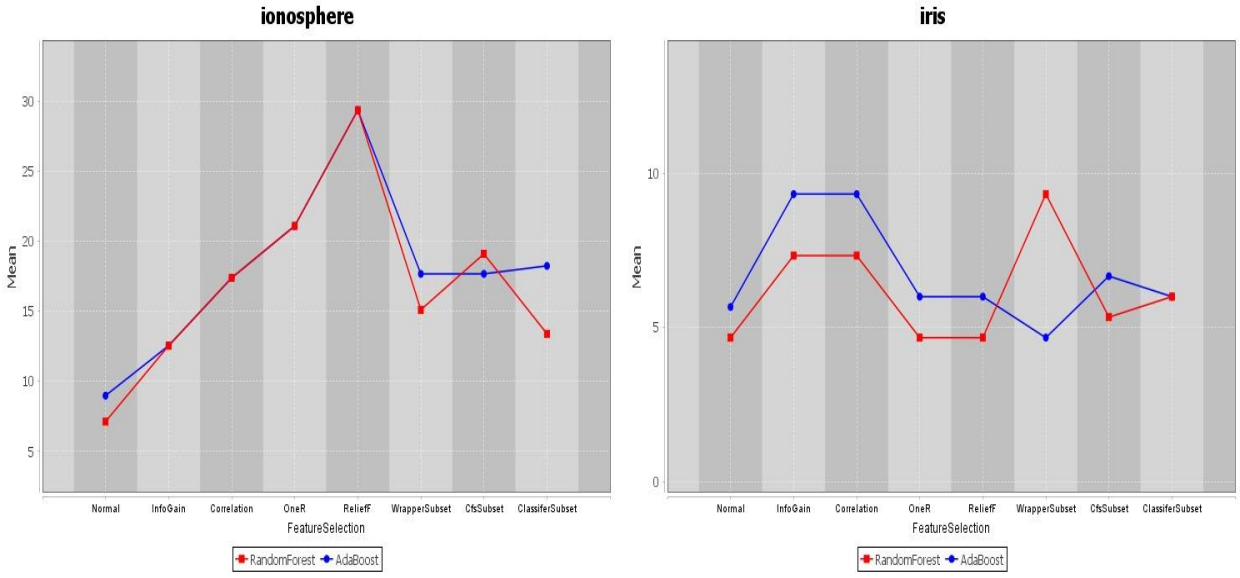

Figura 4-6. Gráfica comparativa de ionosphere e iris.

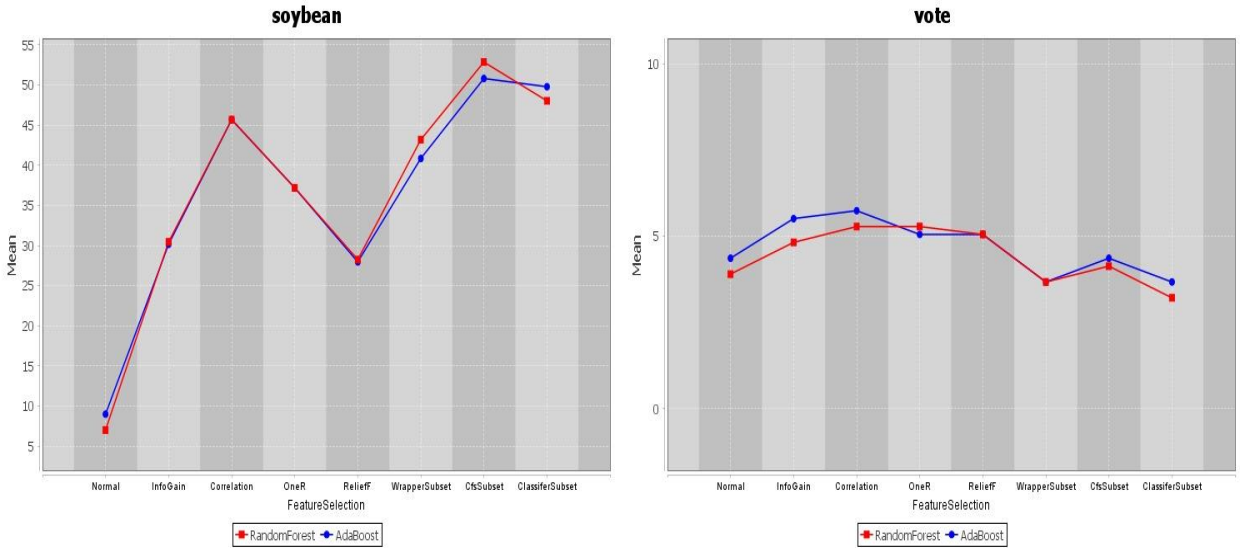

Figura 4-7. Gráfica comparativa de soybean y vote.

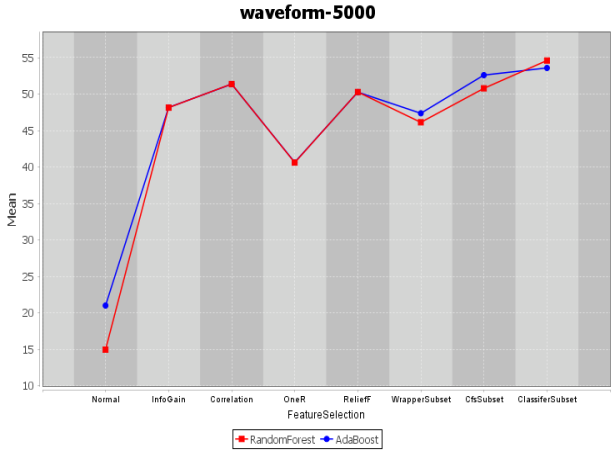

Figura 4-8. Gráfica comparativa de waveform-5000.

Posteriormente, mostramos una tabla detallada con los diferentes porcentajes de error obtenidos que se muestra en la Tabla 4-1.

	Breast-cancer	Credit-g	Diabetes	Glass	Ionosphere	Iris	Soybean	Vote	Waveform-5000
Normal	30,42	23,60	24,35	20,09	7,12	4,67	7,03	3,91	14,94
InfoGain	31,12	28,20	28,39	48,60	12,54	7,33	30,45	4,83	48,14
Correlation	29,37	34,70	28,39	51,40	17,38	7,33	45,68	5,29	51,18
OneR	33,57	35,10	28,39	50,47	21,08	4,67	37,19	5,29	40,60
ReliefF	31,82	30,90	28,39	51,87	29,34	4,67	28,26	5,06	50,28
Wrapper	24,48	31,80	26,17	47,20	15,10	7,33	43,19	3,68	46,12
Cfs	31,12	28,00	26,30	52,80	19,09	5,33	52,86	4,14	50,78
Classifier	29,37	32,90	27,08	53,74	13,39	6,00	48,02	3,22	54,60

Tabla 4-1. Comparativa de errores con diferentes filtros de selección de atributos.

Se puede observar como ninguna de las selecciones de atributos utilizadas nos proporcionan una gran mejora en las estimaciones de error. De hecho, como podemos ver en los conjuntos de datos con gran número de atributos e información, tales como ionosphere, soybean o waveform, las estimaciones de error son paupérrimas utilizando nuestras modificaciones y los filtros de selección de atributos. Esto se debe a que si la selección de atributos no se hace correctamente, se puede dar el caso de estar prescindiendo de los mejores atributos en los nodos con menor profundidad, los cuales son los de mayor importancia, por lo que el árbol no podrá acceder a ellos hasta que la selección de atributos nos lo indique y empeorará drásticamente su precisión. En cambio, si observamos los conjuntos de datos pequeños, como el iris o breast-cancer, obtenemos unos resultados más parejos, debido a que si el filtro de selección de atributos no acierta con el orden de los atributos, no tendrá tanto impacto debido al pequeño tamaño de los árboles resultantes.

4.2.2 Subconjuntos de atributos

En este caso, observaremos el comportamiento de los diferentes algoritmos cuando se elimina la información de sus atributos de forma progresiva.
Para ello, por cada partición de la validación cruzada que se realice, pondremos los valores de un atributo a valor desconocido ("?").
El orden en el cual se eliminarán los atributos vendrá dado por el orden inverso al obtenido en los diferentes filtros de selección de atributos. En el caso de que no se utilicen filtros de selección de atributos, se ha decidido eliminar en el orden normal de los atributos, es decir, si un conjunto de datos tiene N atributos, se empezará eliminando el atributo que está en primer lugar y se acabará eliminando el atributo N en último lugar.

4.2.2.1.1 Subconjuntos de atributos con RandomTree original y sin filtros de selección de atributos

En este caso, se ha experimentado utilizando el RandomTree original sin ningún filtro de selección de atributos. Para ello, utilizaremos un orden de eliminación de atributos en la cual eliminaremos los atributos en el orden en el cual vengan dichos atributos de cada conjunto de datos, es decir, si un conjunto de datos tiene N atributos, eliminaremos el atributo en primera posición en la primera partición, el segundo en la siguiente partición hasta llegar al N-esimo atributo.
En la figura 4-9, se puede observar el comportamiento tanto del RandomForest como del AdaBoost sin modificar cuando se eliminan progresivamente cada uno de sus atributos.

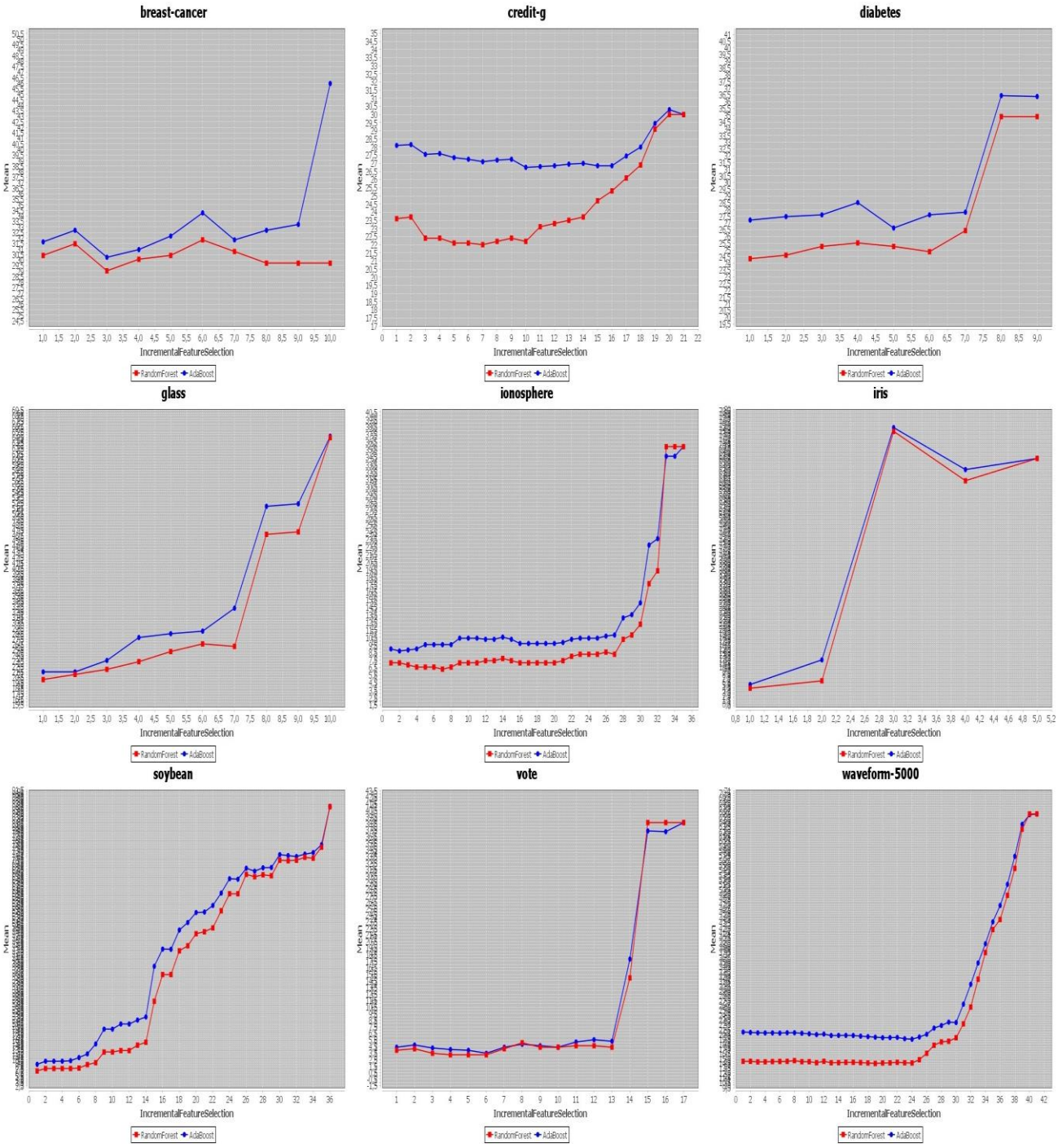

Figura 4-9. Comportamiento del error con eliminación progresiva de atributos.

Podemos analizar estas gráficas de una forma clara observando el comportamiento de los conjuntos de datos iris, soybean y waveform. En el primero, podemos ver una curva muy irregular por lo que suponemos que el primer atributo eliminado apenas le afecta, mientras que el segundo aumenta el error de una forma muy brusca. Esto es debido a una combinación, en la cual el conjunto de datos tiene muy pocos atributos, por lo que no tiene apenas información, y que se ha eliminado el atributo más importante.

En segundo lugar, podemos observar el soybean, en el cual podemos ver una curva creciente relativamente constante, por lo que suponemos que todos los atributos aportan una información similar al conjunto de datos.
Por último, tenemos el waveform, en el que, al contrario que el anterior, no todos los atributos poseen la misma importancia, debido a que como podemos ver en el principio de la gráfica no hay apenas movimiento en el error, hasta que aproximadamente a la mitad de la gráfica empieza a aumentar de una forma brusca, por lo que se ha dado la casualidad de que los atributos importantes son los últimos del orden de eliminación utilizado.

4.2.2.1.2 Subconjuntos de atributos con el RandomTree sin modificar y con filtros de selección de atributos

En segundo lugar, en las figuras 4-10, 4-11, 4-12 y 4-13, mostraremos los resultados en un experimento con el RandomTree sin modificar utilizando los filtros de selección de atributos a la hora de elegir el orden de eliminación de información de atributos, de tal forma que dicho orden será el inverso al obtenido por medio de los filtros.

InfoGain

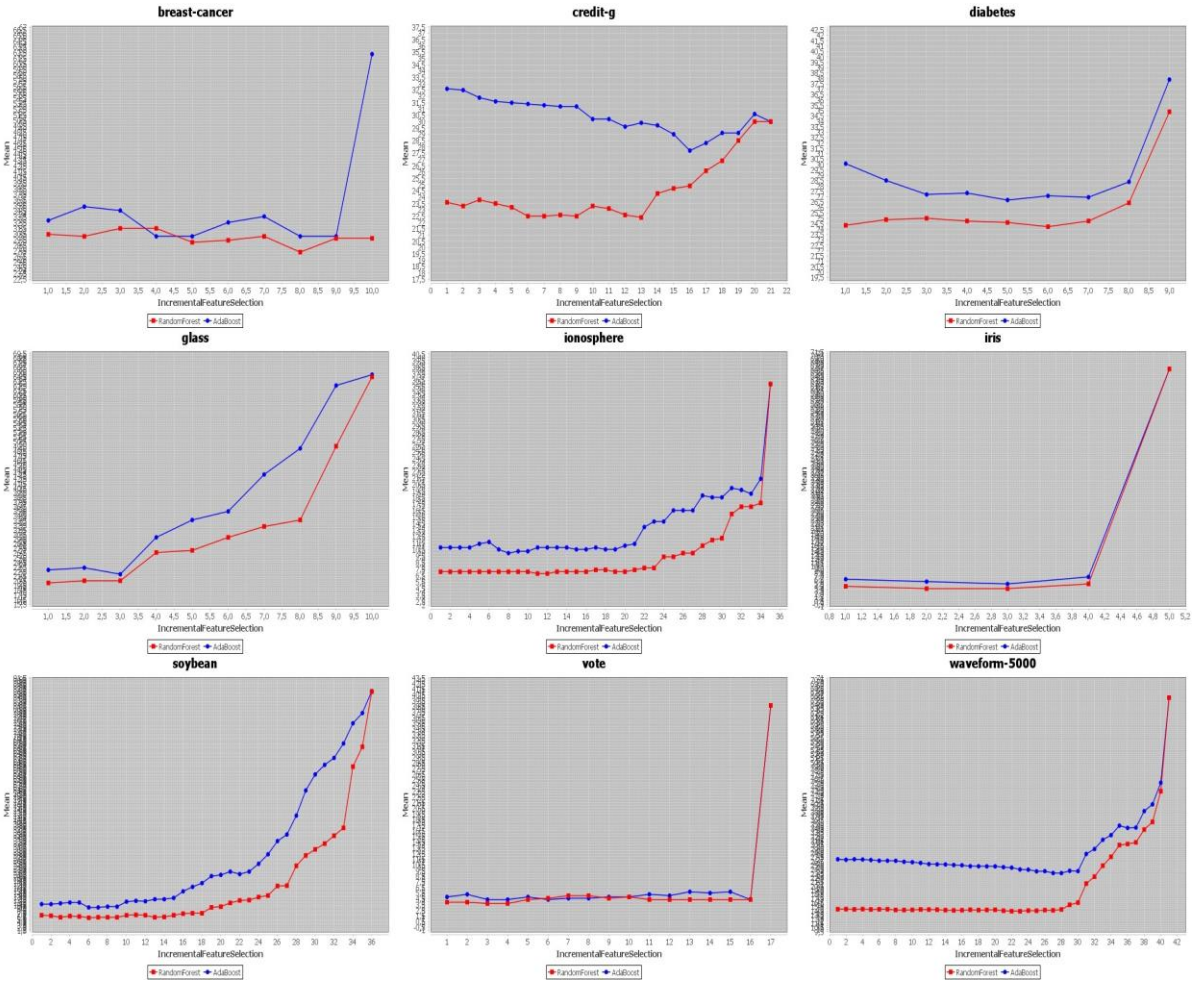

Figura 4-10. Eliminación de atributos con RandomTree sin modificar y filtro InfoGain.

Correlation

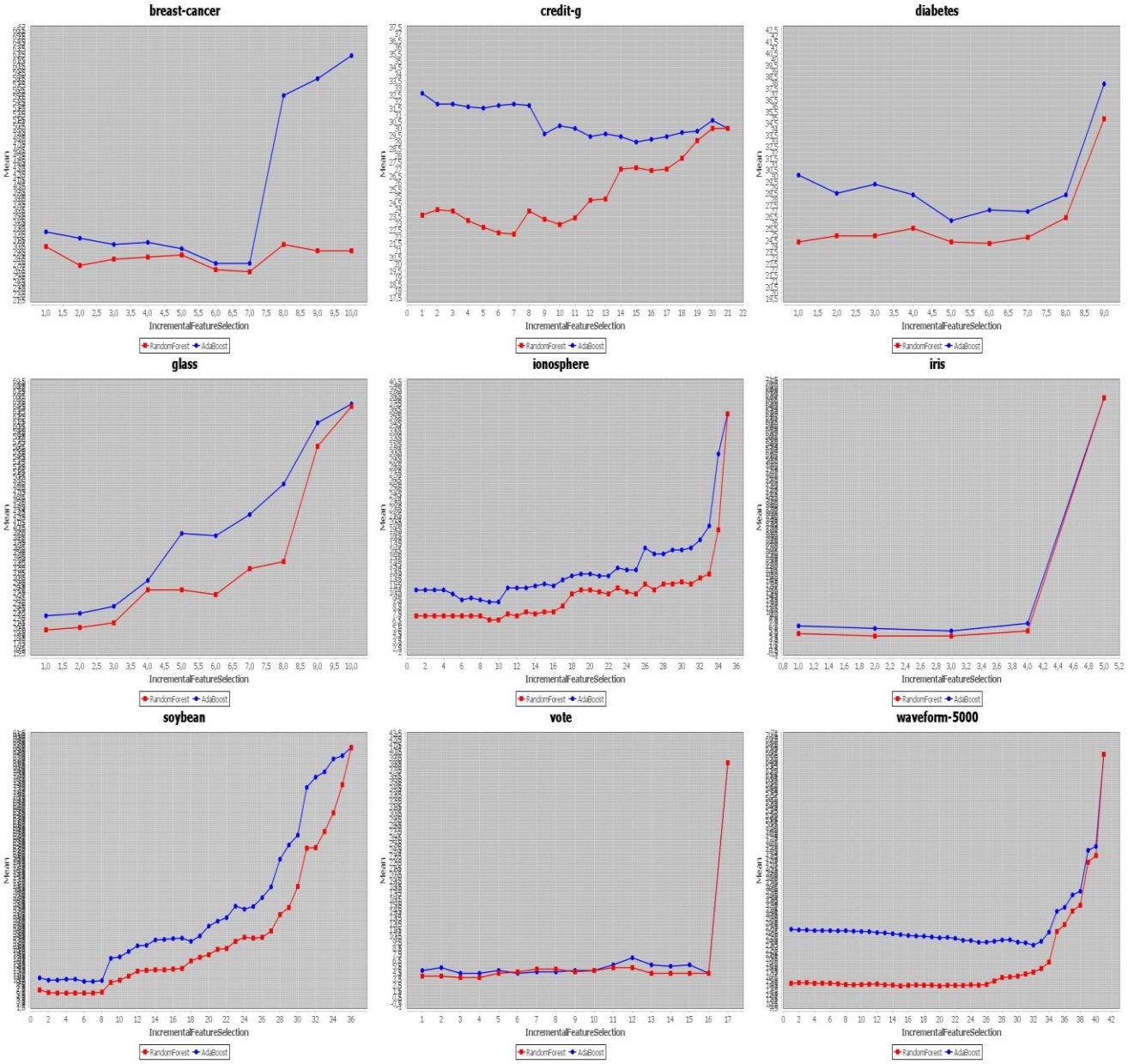

Figura 4-11. Eliminación de atributos con RandomTree sin modificar y filtro Correlation.

OneR

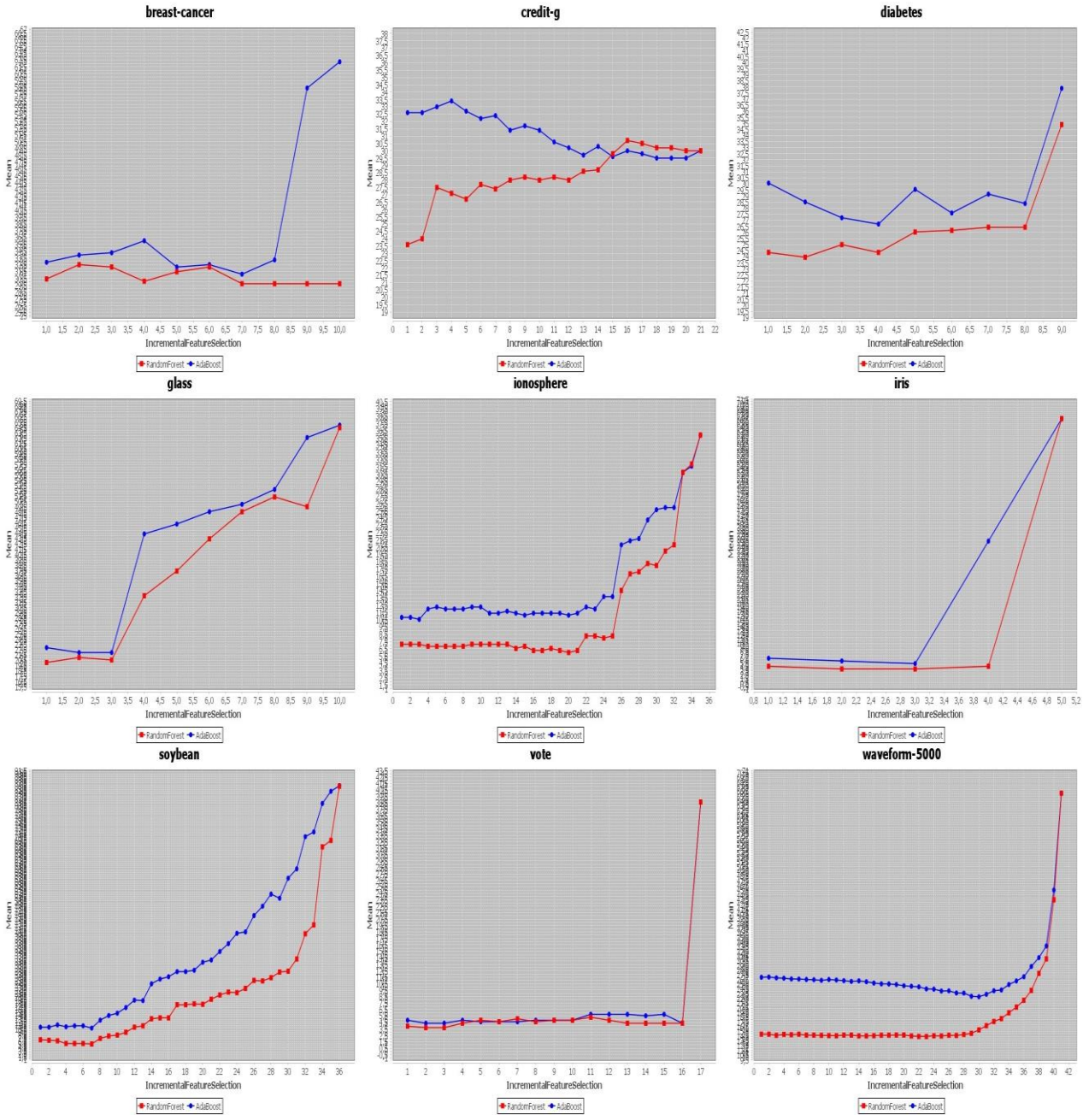

Figura 4-12. Eliminación de atributos con RandomTree sin modificar y filtro OneR.

ReliefF

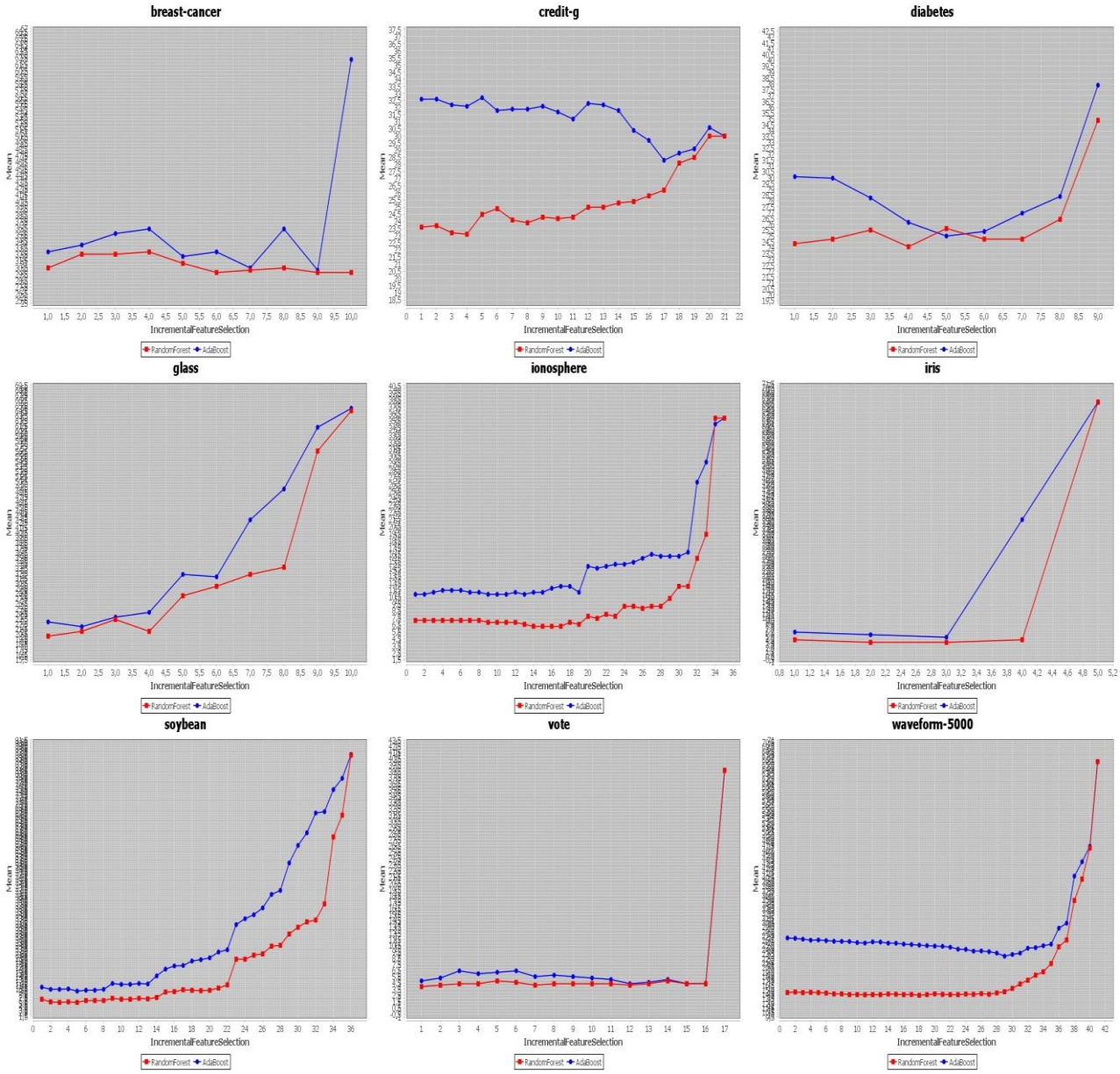

Figura 4-13. Eliminación de atributos con RandomTree sin modificar y filtro ReliefF.

En este caso, podemos observar un comportamiento bastante similar al anterior, pero con la característica de que funciona de una forma bastante más efectiva en los conjuntos de datos pequeños, debido a que esta vez los filtros de selección de atributos nos están indicando que atributos son los que menos impacto van a tener a la hora de eliminarlos, al contrario que en el caso anterior que los eliminábamos de una forma ordenada muy simple y aleatoria.

4.2.2.1.3 Subconjuntos de atributos con el RandomTree modificado y con filtros de selección de atributos

Por último, en las figuras, 4-14, 4-15, 4-16 y 4-17, se muestran los resultados cuando se utiliza el RandomTree modificado junto con los filtros de selección de atributos. Al igual que en el anterior experimento, el filtro también será utilizado de forma inversa a la hora de elegir en qué orden se borrarán los atributos.

InfoGain

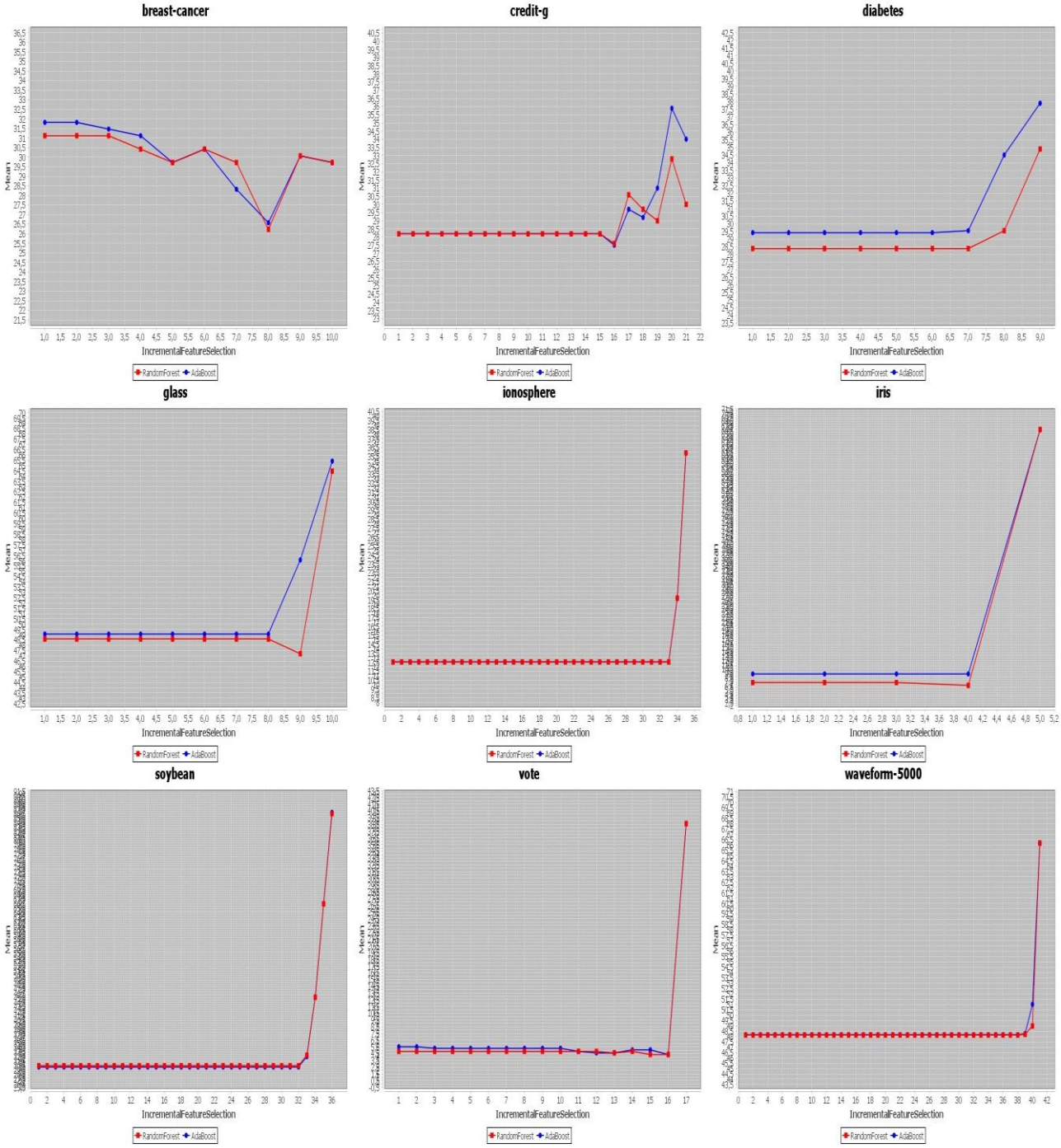

Figura 4-14. Eliminación de atributos con RandomTree modificado y filtro InfoGain.

Correlation

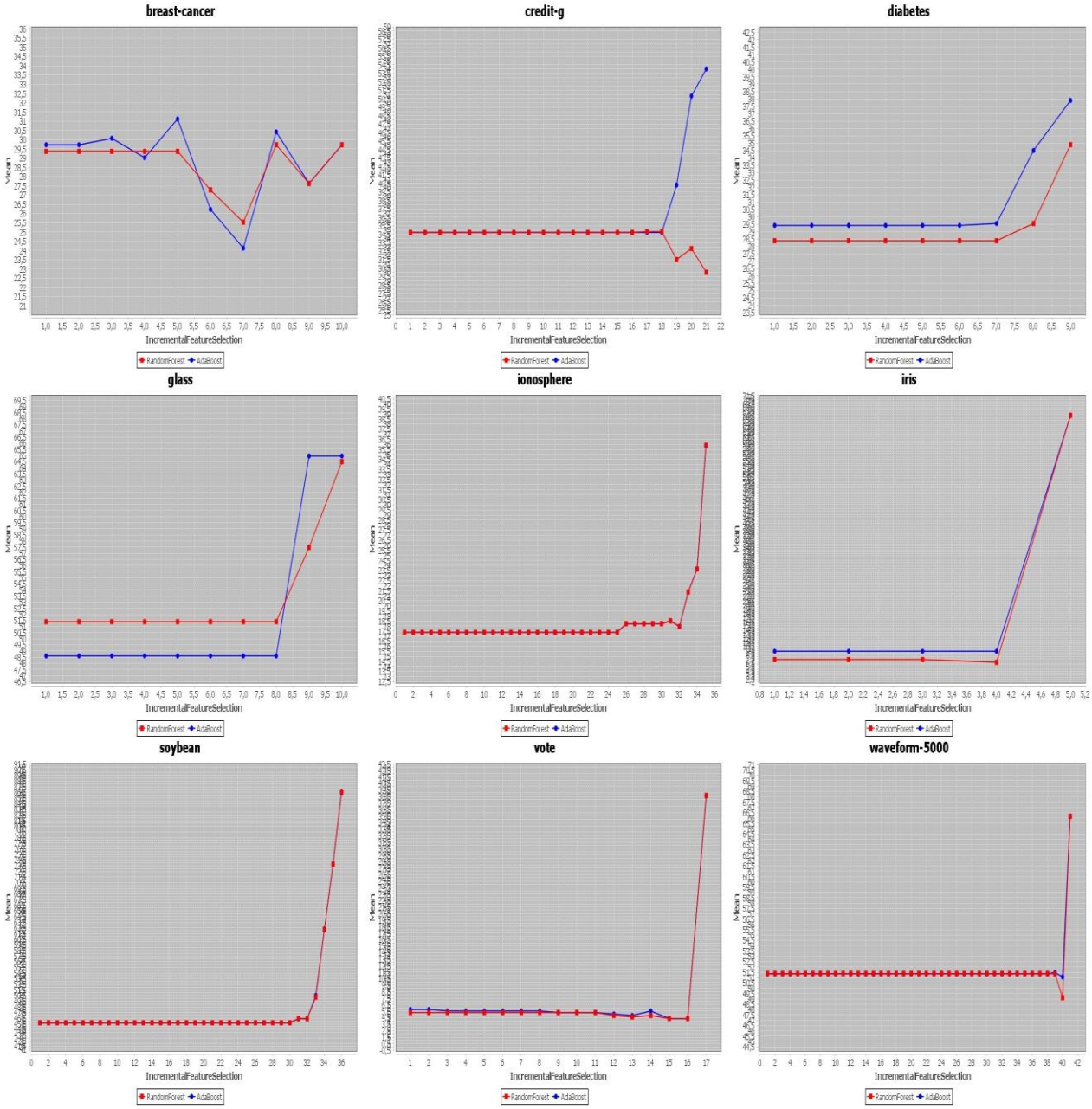

Figura 4-15. Eliminación de atributos con RandomTree modificado y filtro Correlation

OneR

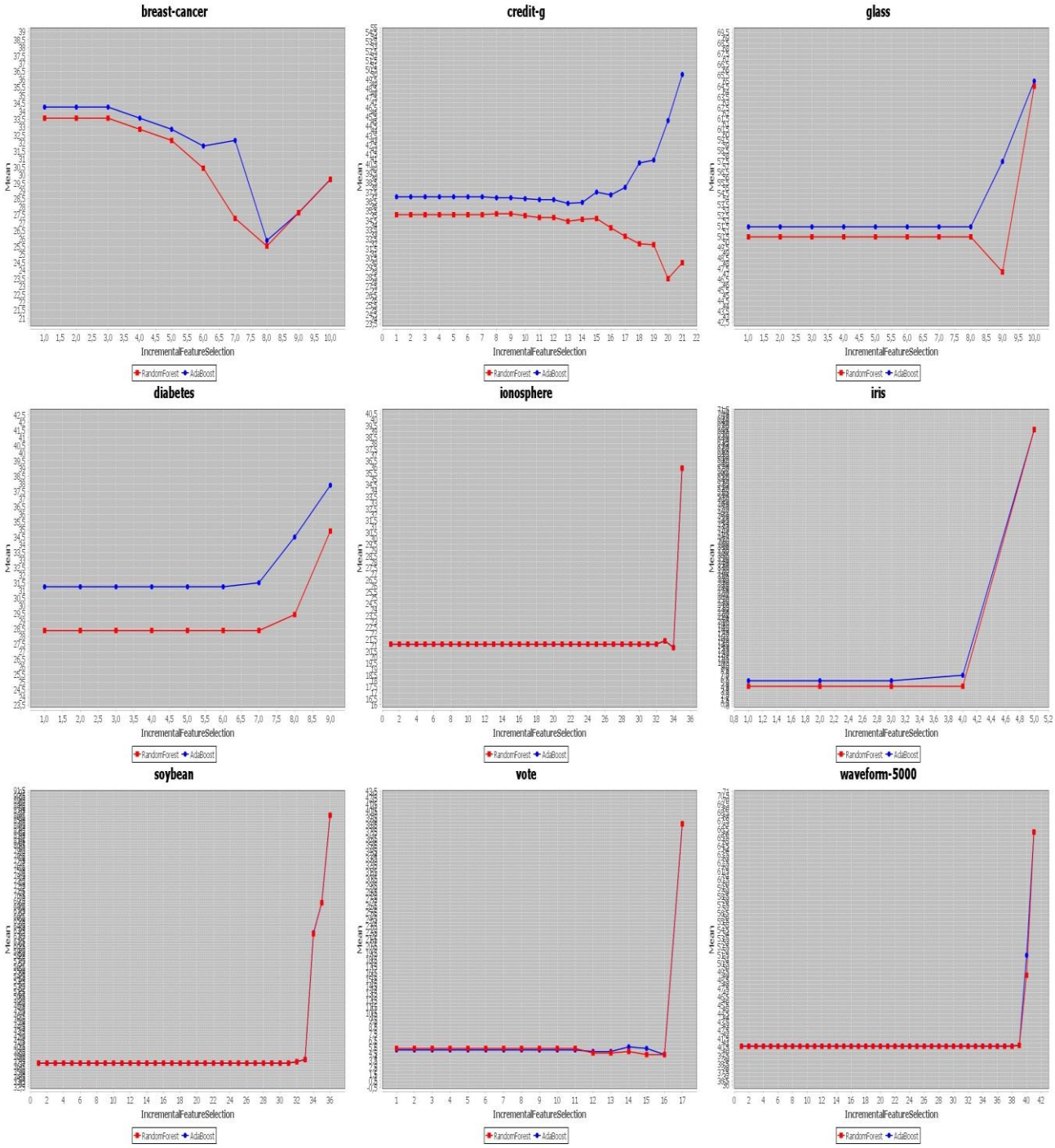

Figura 4-16. Eliminación de atributos con RandomTree modificado y filtro OneR.

ReliefF

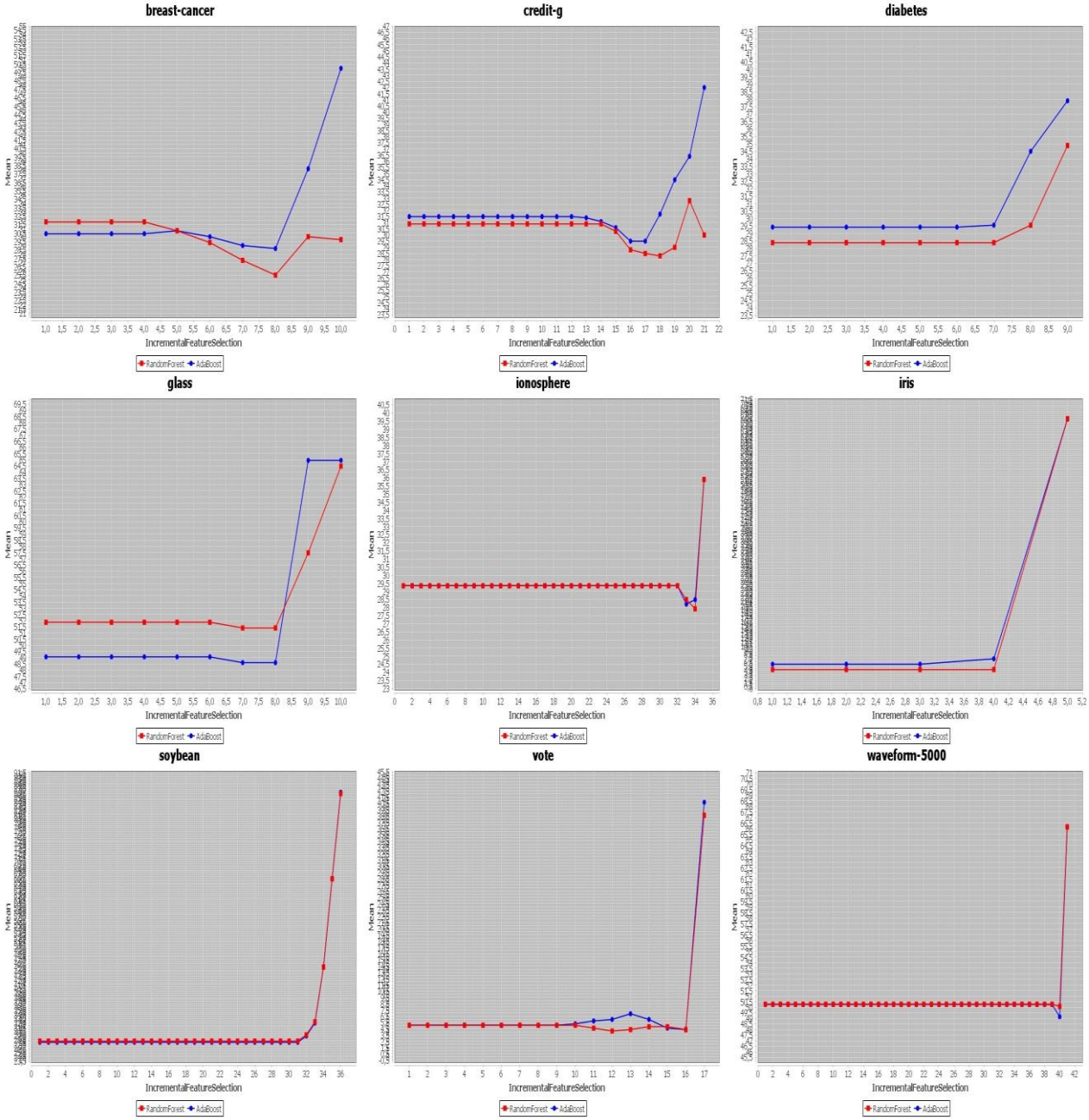

Figura 4-17. Eliminación de atributos con RandomTree modificado y filtro ReliefF.

Finalmente, podemos analizar este último caso experimentado, en el cual también se elimina la información de cada atributo pero utilizando las modificaciones de los algoritmos de clasificación. En ellas, vemos que la mayoría de los filtros de selecciones de atributos predicen bastante bien a la hora de clasificar los conjuntos de datos que menos atributos ofrecen, ya que se observa como el error no se inmuta hasta que empezamos a llegar a los últimos atributos en los cuales se producen modificaciones en la curva drásticas según el conjunto de datos o el filtro utilizado.
Esto nos lleva a la conclusión de que los filtros tienen una buena capacidad a la hora de descartar atributos que no aportan información.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

En este Trabajo de Fin de Grado se ha estudiado el comportamiento de algoritmos del entorno del aprendizaje automático, con la finalidad de obtener mejores resultados en cuanto los resultados de predicción y tiempos de trabajo.

Para ello, se ha estudiado el comportamiento de algoritmo boosting en comparativa con el RandomForest, aplicándoles a cada uno de ellos unas pequeñas modificaciones en código interno.

Adicionalmente, se han aplicado los filtros de selección de atributos a dichos algoritmos con la intención de qué los subconjuntos de atributos utilizados fuesen los mejores posibles.

Tras los experimentos realizados, podemos concluir en que no hemos obtenido grandes mejoras o rendimientos de los diferentes algoritmos modificados, pero que sí hemos obtenido información de bastante utilidad, como, por ejemplo, que las modificaciones afectan de la misma manera tanto al Adaboost como al Random Forest o que los filtros de selección de atributos son bastante efectivos a la hora de descartar atributos, pero no igual a la hora de seleccionar los mejores.

5.2 Trabajo futuro

En cuanto al trabajo futuro que podemos desarrollar en torno a estos experimentos llevados a cabo alrededor del boosting y el bagging, tenemos bastante material en el que poder trabajar.

Por una parte, podemos partir por el lado de los filtros de selección de atributos, en los cuales hemos visto una gran eficacia en los peores atributos pero una gran inestabilidad en los mejores, por lo que podemos hacer diferentes experimentos o pruebas en torno a esa parte de dichos filtros con el objetivo de mejorarlos.

Por otro lado, podemos retocar las modificaciones de nuestro particular RandomTree de forma que, aunque la selección de atributos haya sido inestable o fallida, no conlleve un rendimiento tan pobre en los resultados de los conjuntos de datos más grandes.

Referencias

[1] F. Sancho (2017). Clasificación supervisada y no supervisada. Universidad de Sevilla.

[2] B. Hssina, A. Merbouha, H. Ezzikouri, M.Erritali. A comparative study of decision tree ID3 and C4.5. TIAD Laboratory, Sultan Moulay Slimane University.

[3] L. Breiman (1996). Machine learning, volume 24, page 123. Kluwer Academic Publishers.

[4] R.A. Berk (2008). Random Forests, Statistical Learning from a Regression Perspective. Springer, Cham.

[5] R.A. Berk (2008). Boosting, Statistical Learning from a Regression Perspective. Springer, Cham.

[6] L. Reyzin (2010). Boosting on a Budget: Sampling for Feature-Efficient Prediction. Georgia Institute of Technology.

[7] G. Martínez (2006). Clasificación mediante conjuntos. Universidad Autónoma de Madrid.

[8] Y. Freund (1995). Information and Computation, volume 121, issue 2, pages 256-285. AT&T Bell Labs, Murray Hill.

[9] L. Breiman (2001). Machine learning, volume 45, issue 1, pages 5-32. Kluwer Academic Publishers.